Eur. Phys. J. C (2015) 75:467
DOI 10.1140/epjc/s10052-015-3638-8

THE EUROPEAN PHYSICAL JOURNAL C

Special Article - Tools for Experiment and Theory

PySLHA: a Pythonic interface to SUSY Les Houches Accord data

Andy Buckley[a]

School of Physics and Astronomy, University of Glasgow, Glasgow, UK

Received: 16 June 2015 / Accepted: 17 August 2015

Abstract This paper describes the PySLHA package, a Python language module and program collection for reading, writing and visualising SUSY model data in the SLHA format. PySLHA can read and write SLHA data in a very general way, including the official SLHA2 extension and user customisations, and with arbitrarily deep indexing of data block entries and a dedicated, intuitive interface for particle data and decay information. The draft SLHA3 `XSECTION` feature is also fully supported. PySLHA can additionally read and write the legacy ISAWIG model format, and provides format conversion scripts. A publication-quality mass spectrum and decay chain plotting tool, `slhaplot`, is included in the package.

1 Introduction

The SUSY Les Houches Accord (SLHA) data format [1] has proven an effective and exceedingly popular mechanism for exchange of weak-scale supersymmetry model information between spectrum generator codes (e.g. SoftSUSY [2], SPheno [3], and many others) to Monte Carlo event generators and other users of weak scale SUSY parameters. As such, SLHA has become the current de facto format for exchange of SUSY model data, replacing various code-specific formats such as the ISAWIG format [4] used to exchange ISAJET-generated [5] model data to the FORTRAN HERWIG event generator [6].

SLHA's text-based format was designed to be easily written and read by hand, and by the prevalent FORTRAN-based spectrum generators and MC codes at the time of its definition. The original format was defined during and following the Les Houches 2003 workshop [1], and was extended in 2007 to the current version 2 of the SLHA standard [7,8]. The format consists of *blocks* of model data, each identified by a header line starting with either `BLOCK` or `DECAY` for model data blocks and particle decays respectively. Each data block has a name and an associated energy scale at which it is defined, as well as many *entries*. Decay blocks each define the total decay width and a list of constituent decay channels and branching ratios for a particle species (identified using the PDG MC numbering scheme). The latest update of the SLHA standard introduces a `XSECTION` block type for representation of calculated process cross-sections at various energies and in different calculation schemes.

Several SLHA reader codes already exist in the FORTRAN and C++ languages, but the popular Python language was until recently unblessed with such functionality. Python is an interpreted language, particularly well suited to clear and effective expression of complex logic, and particularly has advantages over FORTRAN, C++ and other numerical/system programming languages where "high-level" operations such as rich string parsing are involved. This is often the case in SUSY model exploration, where Python can act as a glue to tie together the input and output stages of several distinct modelling programs. The PySLHA library described in this note is a Python language module (and several Python programs) for reading, writing, and converting SLHA model data, as well as providing a high quality plotter of SUSY model mass spectra and decay chains, hence providing a convenient and powerful integration of SLHA into the Python "glue layer" for BSM model studies.

2 Library features

PySLHA's main programmatic features are the `Doc`, `Block`, `Particle` & `Decay`, and `Process` & `XSec` classes, which map directly on to features of the SLHA data format. We

[a] e-mail: andy.buckley@cern.ch

Published online: 01 October 2015

first describe these structures and then the functions which operate on them.

2.1 **Doc**

From PySLHA 3.0.0, the main reading and writing functions in PySLHA accept and return a single **Doc** object; before this, they passed several **dict**s of data objects, but the single **Doc** is more robust and extensible. **Doc** is a simple structured container which provides access to the main groups of SLHA data types, via its **blocks**, **decays** and **xsections** attributes. A **write** method bound to the **Doc** allows convenient file and string I/O of its data, with controllable precision, in addition to the unbound **write*** functions described in Sect. 2.6.

A SLHA document can be accessed as follows:

```
>>> import pyslha
>>> d = pyslha.read("myspectrum.slha")
>>> print d
<PySLHA Doc: 16 blocks, 32 decays, 94 xsections>
```

In future versions, the **Doc** (and the following types) will allow storage of SLHA document inline and block comments as well as the parsed data. For now, only non-comment data will be preserved in a read/write cycle.

If it can be imported, the **OrderedDict** Python type will be used as a base type for the **blocks**, **decays** and **xsections** containers, which have a **dict**-like interface, in order to ensure that the user-specified ordering of data sections in the SLHA file is not disrupted by manipulation in PySLHA.

2.2 **Block**

From PySLHA 2.0.0 onward, the **Block** class provides a similar interface to Python's native **dict** type, i.e. an instance of **Block** contains several entries accessed by an index or key, either by named functions or via the square-brackets `[x]` operator. **Block**s also have an associated name and a Q value indicating the scale at which the contents are defined.

Like **dict**s, **Block** instances may store any type of object as the value of an entry, although the SLHA file format means that in practice this is restricted to integers, floating point numbers, and character strings (and sequences of the above). Also like **dict**s, **Block**s support the Python iterator protocol, the get/set-item syntax, and methods such as **items()**, **values()**, **keys()**, and **has_key()**. *Unlike* **dict**s, **Block** entries can only be indexed on integers, or on tuples of integers – or, in a special case designed for handling the standard **ALPHA** block, a block may contain a single unkeyed entry. The ability to use tuples of integers as keys is particularly useful for blocks which represent mixing matrices, e.g.

```
umix = pyslha.Block("UMIX", q=1e7)
umix[1,1] = 0.1
...
print umix[1,1]

mymix = pyslha.Block("MYMIX")
mymix[2,1,3] = 3.142
...
print mymix[2,1,3]
```

Adding and accessing multi-value block entries by integer (tuple) keys is unambiguous when done programatically as above. PySLHA of course also has to be able to parse SLHA entry lines into blocks and here some heuristics are necessary to differentiate between keys and values. The approach adopted, via the **add_entry** and **set_value** methods, is that if the supplied argument is a string, it will be split into a tuple of strings; these are then automatically converted to numeric types if possible, and consecutive strings are combined together (to allow for string values containing spaces.) The resulting tuple of ints, floats and strings is then parsed from left to right to find the first non-int item: all items before this are treated as the key.

A strength of Python as a language for SLHA block parsing is that the language is very dynamically typed: a variable can hold any type, and **dict**s, **tuple**s, and other containers can hold heterogeneous entries. This is what makes it possible for blocks to be indexed by integer tuples of any length (or no length) and for values to either be scalar or tuples of mixed types. Hence no special features are needed for SLHA2 support or further extensions to the SLHA standard block content: any block layout representable in the SLHA syntax (and some which are not) can be manipulated using the PySLHA **Block** class.

2.3 **Particle** and **Decay**

The **Particle** and **Decay** classes are the complements of **Block** for SUSY particle properties and decay specifications. Particles are defined by a PDG species identifier code [9] and a list of **Decay**s, and optionally may also contain the total decay width and the mass[1] in GeV. In this sense, **Particle**s are representations of the SLHA **DECAY** block itself, while its entries are each transformed into a fully-fledged **Decay** object. Each **Decay** contains a branching ratio and a list of particle ID codes representing the decay daughters. For convenience and familiarity, the PDG ID of the decaying particle and the **NDA** (number of daughters) number may also be stored but these are not essential: the parent particle has its own fully fledged **Particle**

[1] There is intentional duplication here: the SLHA format design places particle masses in the **MASS** block, but from a code design point of view it makes sense for a **Particle** to know its own mass.

instance and **decay.nda** is equivalent to the "more Pythonic" **len(decay.ids)**.

2.4 Process and XSec

The SLHA3 draft standard[2] introduces a new block type, the **XSECTION**, for storing various calculated cross-sections for different processes. This is supported in PySLHA from version 3.1.0 onwards, via the **Process** and **XSec** types. As for the **DECAY** block, a direct representation of the SLHA text format in memory is not the most natural programming interface, and hence a slightly different structure is used.

The **Process** class is the closest match to an **XSECTION** block, and the **XSec** objects contained within it map to the entries in the block. The main distinction is that a different **XSECTION** block is used for each energy at which a process cross-section is provided, which the **Process** is a **dict**-like object keyed on the concatenated tuple of (sorted) initial- and final-state particle ID codes. A given **Process** can then contain **XSec** entries at several energies, as well as the variations in QCD and EW order, factorization and renormalization scale-factors, PDFs, and the computational code which made the calculation. This is both more semantically natural, and avoids the problem of indexing the dictionary on $\sqrt{s}$, a floating-point number which could easily fail to compare exactly equal to the required key value.

The **XSec** objects inside a **Process** are available in the **xsecs** list attribute. To make look-up of cross-sections which meet certain criteria easier, however, a filtering function has been provided which only requires a subset of the scheme details, e.g. the energy, renormalization scale and code, to be specified. The following example demonstrates how to find the available process index tuples and query them – in this case for a spectrum file which contains cross-sections at 8 TeV but none at 13 TeV:

```
>>> d = pyslha.read("myspectrum.slha")
>>> d.xsections.keys()
[(2212, 2212, 1000001, 1000003),
 (2212, 2212, -1000002, 2000002), ...]
>>> proc = d.xsections[2212, 2212, 1000001, 1000003]
>>> proc.get_xsecs(sqrts=13000., kappa_r=2)
[sqrt(s) = 8000 GeV, avg mass scale scheme, ...]
>>> proc.get_xsecs(sqrts=13000)
[]
```

2.5 String representation

Block, **Particle** and **Decay** objects can all represent themselves in convenient string form, as shown here for the NMIX matrix, neutralino LSP and gluino entries read from an example mSUGRA file (and accessed via two **dict**s – more on this later):

```
>>> print blocks["NMIX"]
NMIX { 1,1 : 0.94321758; 1,2 : -0.20379469; ...
>>> print decays[1000022]
1000022 : mass = 9.66880686e+01 GeV :
    total width = 0.00000000e+00 GeV
>>> print decays[1000021]
1000021 : mass = 6.07713704e+02 GeV :
    total width = 5.50675438e+00 GeV
1.05840237e-01 [1000005, -5]
...
```

2.6 File and string I/O

The objects previously described are the in-memory data which can be programmatically manipulated by the user. To read these objects from SLHA files, the functions **readSLHA()** and **readSLHAFile()** are supplied.

The format parsing itself lives in the former, which takes an SLHA file's content as a string argument, and returns two **dict**s: one containing the blocks keyed by name, and the the other containing the **Particle** objects keyed by PDG ID code.[3] The second ("**File**") function treats its argument as either a filename string or as a Python file object, from which it loads the file content and passes it to **readSLHA()**. Both forms take an optional argument, **ignorenobr**, which if set true will exclude any **Decay** objects with a branching ratio of 0 from the resulting **Particle**s; it is set false by default. Usage of these functions is demonstrated here:

```
doc = pyslha.readSLHA(myslhastring)
...
doc = pyslha.readSLHAFile("sps1a.slha",
                          ignorenobr=True)
```

SLHA format writing from code objects is similarly simple. Having placed **Block** and **Particle** objects (the latter containing **Decay**s) into **dict**s, cf. the return values of the reader functions, they are passed to symmetric writer functions as follows:

```
slhastring = pyslha.writeSLHA(doc,
                              ignorenobr=True)
...
pyslha.writeSLHAFile("mymodel.slha", doc,
                     precision=6)
```

The optional **precision** argument specifies how many decimal places to be written for floating point values such as masses, branching ratios, and widths: the default is 8. Two further functions, **writeSLHABlocks(blocks)** and **writeSLHADecays(decays)** exist to separately produce the

[2] https://phystev.cnrs.fr/wiki/2013:groups:tools:slha.

[3] Again, these will be *ordered* **dict**s if possible, ensuring that the order of blocks when iterated/written out matches that in which they were read in. The same applies to entries within blocks.

SLHA output strings for the blocks and decays collections: both accept an optional **`precision`** argument, and the optional **`ignorenobr`** argument may be passed to the decay writer.

2.7 HERWIG/ISAWIG ↔ SLHA conversion

An original major motivation for PySLHA was to convert SLHA spectrum/decay files to the format used by the **`Fortran`** HERWIG event generator [4,6] for SUSY simulation. This format was previously only output by the ISAWIG program, which uses the SUSY spectrum generator tools from the ISAJET code [5]. The restriction of HERWIG SUSY simulation to spectra generated by ISAJET/ISASUSY, and the increasing difficulty of building ISAJET (due to reduced availability of CERNLIB and Patchy, and increased standard enforcement in standard Fortran compilers) motivated development of a format converter which would permit other spectrum generators to produce HERWIG-compatible spectrum files.

Although use of ISASUSY (and of HERWIG) has reduced in recent years, PySLHA retains the ability to read and write the ISAWIG format. The **`readISAWIG()`** and **`readISAWIGFile()`** will parse the ISAWIG format into the PySLHA objects cf. the **`readSLHA*()`** ones, and the **`writeISAWIG()`** and **`writeISAWIGFile()`** functions invert the process. Unlike SLHA, in which the data block format is arbitrarily extensible, the ISAWIG format is fixed and only a subset of data will be written out: the corresponding block entries *must* be present.

There are some incompletenesses in PySLHA's ISAWIG support due to a requirement that some SUSY decays be ordered in the file according to their dependence on HERWIG internal matrix elements: it is not clear that these can be handled in full detail without linking against HERWIG. Additionally, R-parity violating couplings are not currently read in or written. These defects will be happily resolved if possible and if there is sufficient interest.

Conversion between the ISAWIG and SLHA formats is made easier by two simple converter scripts, **`isawig2slha`** and **`slha2isawig`**, so that programming in Python is not necessary to convert a file of one sort into the other.

A side effect of having the ISAWIG ↔ SLHA converter machinery in PySLHA is that the library contains two functions, **`herwigid2pdgid(hwid)`** and **`pdgid2herwigid(pdgid)`**, which as their names suggest convert particle ID codes between the standard PDG scheme and the HERWIG internal scheme. This may be useful, although undoubtedly less so as a new generation of C++ event generator codes replace the venerable **`Fortran`** ones.

3 Mass spectrum and decay visualisation with `slhaplot`

One of the most heavily used features of PySLHA, and a large part of the original motivation for the package, is the **`slhaplot`** script, which produces high quality graphical representations of SUSY mass spectra and decay chains in a variety of formats including PDF, EPS, and LaTeX. As usual, the particle species are separated into distinct columns for Higgses, sleptons, gauginos, and squarks/gluinos: an example is shown in Fig. 1.

Use of **`slhaplot`** is simple: for default output a number of SLHA (or ISAWIG) filenames are supplied on the command line and each will produce a resulting PDF plot (or plots)

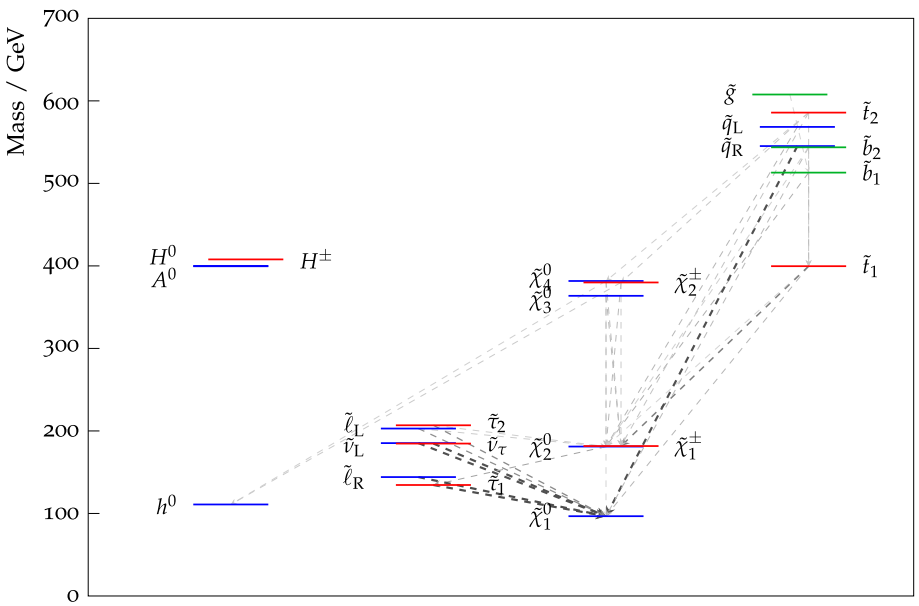

Fig. 1 An example of the mass spectrum/decay channel visualisation output from **`slhaplot`**, showing the now-defunct SPS1 SUSY benchmark point

with the same base name. A **format** flag may be given to specify which output formats to use, as a comma-separated list: more than one is allowed if e.g. output of the same plot in PDF, EPS, PNG and LaTeX formats were to be useful. Fundamentally the plots are produced using the PGF/TikZ [10] TeX-based graphics library, and the **tex** output format contains the raw TikZ programming statements for modification by the end user if wanted. A LaTeX preamble may be supplied to **slhaplot** by users who wish to e.g. use different fonts, and another option allows a TikZ fragment to be included for easy adding of drawing items, e.g. extra labels on the plot.

Many further rendering options are available via the **slhaplot** command line: these include

- a minimum branching ratio to be displayed, to avoid cluttering the plots with virtually non-existent decays;
- a maximum particle mass to be displayed; by default this is calculated from the mass spectrum, and the plot axis extent chosen for cosmetic appropriateness;
- variations on the rendering style for decay arrows (BR-dependent line widths and line colours);
- variations in mass-line labelling algorithms – **slhaplot** makes efforts to avoid ugly and unhelpful overlaps of particle name labels resulting from near-degenerate masses. The user may choose either to merge close-by labels into a single label, or to shift labels by small amounts to avoid the clashes. The latter is the default behaviour and produces good results most of the time: if fine control is needed, the user may dump and modify the LaTeX plot source code;
- changing the plot aspect ratio.

The Python rendering engine which calls **latex**, **pdflatex**, and format converter programs (with caching of intermediate stages for efficiency) has been split off from PySLHA as the separate **tex2pix** package [11].

4 Summary

PySLHA is a mature Python library for reading, writing and manipulating SUSY model data with input and output to the SLHA and ISAWIG formats. Also included are scripts for easy conversion between these two data formats, and for producing publication-ready SUSY mass-spectrum/decay plots.

PySLHA is available from the Python package index (PyPI) at https://pypi.python.org/pypi/pyslha and a home page with more usage instructions is available at http://insectnation.org/projects/pyslha. The code is documented according to normal Python standards which may be queried via **pydoc** or the Python interactive **help()** command. The latest version at the time of writing is PySLHA 3.1.1.

Acknowledgments I started PySLHA for personal amusement and challenge (and with the hope of avoiding future compilations of ISAJET/CERNLIB) at an MCnet summer school in 2010. It was later extended – in particular with the addition of **slhaplot** – for use in graduate lectures on BSM physics organised by the Scottish Universities Physics Alliance (SUPA). My thanks both to MCnet and SUPA for their support via travel funding and a research fellowship respectively. I further acknowledge and thank the Royal Society for their funding of a University Research Fellowship, CERN for a visiting Scientific Associateship, and the Durham University Institute for Particle Physics Phenomenology for several small associateship grants: all these have greatly assisted with collaborative work on particle physics MC modelling, PySLHA included. Thanks also to the many users who made development suggestions (such as writing this paper!) and let me know that PySLHA had active users; without their appreciation and enthusiasm it would have been all too easy to let the project stagnate once it met my own needs.

Funded by SCOAP[3].

References

1. P.Z. Skands, B.C. Allanach, H. Baer, C. Balazs, G. Belanger, F. Boudjema, A. Djouadi, R. Godbole et al., SUSY Les Houches accord: Interfacing SUSY spectrum calculators, decay packages, and event generators. JHEP **0407**, 036 (2004). arXiv:hep-ph/0311123
2. B.C. Allanach, SOFTSUSY: a program for calculating supersymmetric spectra. Comput. Phys. Commun. **143**, 305 (2002). arXiv:hep-ph/0104145
3. W. Porod, SPheno, a program for calculating supersymmetric spectra, SUSY particle decays and SUSY particle production at e+ e- colliders. Comput. Phys. Commun. **153**, 275 (2003). arXiv:hep-ph/0301101
4. P. Richardson, "ISAWIG". http://www.hep.phy.cam.ac.uk/richardn/HERWIG/ISAWIG/
5. F.E. Paige, S.D. Protopopescu, H. Baer, X. Tata, "ISAJET 7.69: a Monte Carlo event generator for pp, anti-pp, and e+ reactions. arXiv:hep-ph/0312045
6. G. Corcella, I.G. Knowles, G. Marchesini, S. Moretti, K. Odagiri, P. Richardson, M.H. Seymour, B.R. Webber, HERWIG 6: an event generator for hadron emission reactions with interfering gluons (including supersymmetric processes). JHEP **0101**, 010 (2001). arXiv:hep-ph/0011363
7. B.C. Allanach, C. Balazs, G. Belanger, M. Bernhardt, F. Boudjema, D. Choudhury, K. Desch, U. Ellwanger et al., SUSY Les Houches Accord 2. Comput. Phys. Commun. **180**, 8 (2009). arXiv:0801.0045 [hep-ph]
8. P. Skands, "Supersymmetry Les Houches Accord website. http://home.fnal.gov/skands/slha/
9. C. Amsler et al. (Particle Data Group Collaboration), Review of particle physics. Phys. Lett. B **667**, 1 (2008) . (MC particle ID scheme, http://pdg.lbl.gov/2007/reviews/montecarlorpp.pdf)
10. T. Tantau, The PGF/TikZ LaTeX package. http://www.ctan.org/pkg/pgf
11. A. Buckley, The tex2pix Python package. https://pypi.python.org/pypi/tex2pix